Heaps

Supply a Comparator to change the behaviour ⇒ **Priority Queue** → in Java / Python

- getMax() → O(1)
- remove Max() → O(Log N)
- insert (d) → O(Log N)

default

- getMin()
- remove Min( )
- insert(d)

Library Implementation

$O(N \log N)$

Concept - Building a Heap from an array

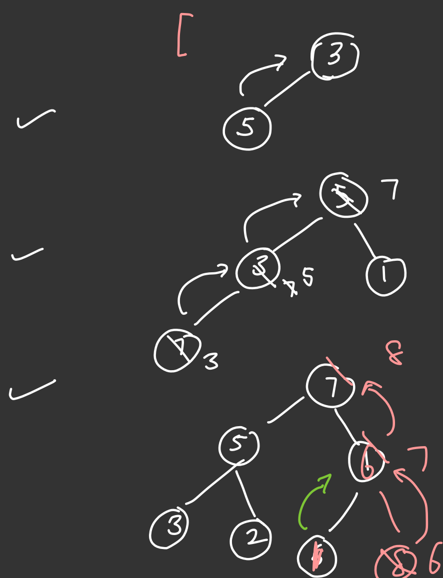
max Heap

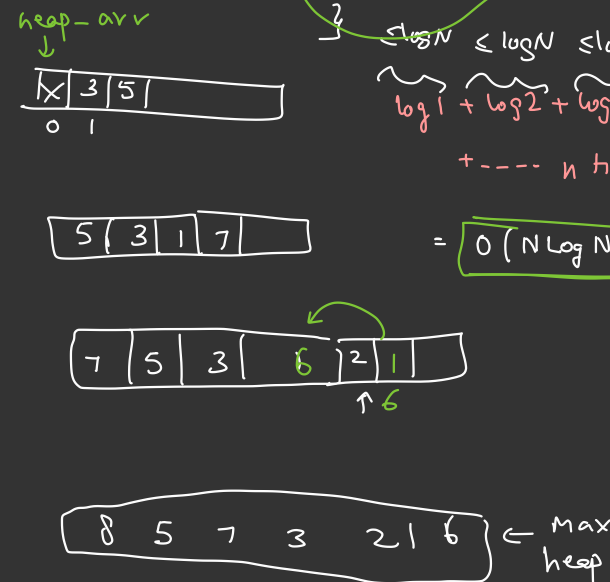
3, 5, 1, 7, 2, 6, 8 ← input

```
for (x: input){
    pq.insert(x)
}
```

$\underbrace{\le \log N}_{\log 1} + \underbrace{\le \log N}_{\log 2} + \underbrace{\le \log N}_{\log 3} + \text{----} \; n \text{ times}$

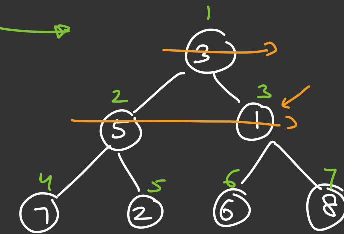
$= \boxed{O(N \log N)}$



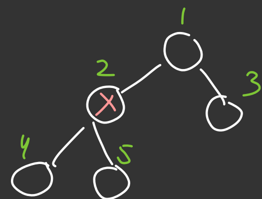


New Problem / Concept → Convert the input array into heap array in place $O(N)$

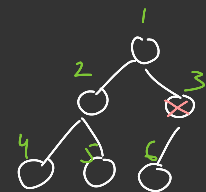
0	1	2	3	4	5	6	7
X	3	5	1	7	2	6	8



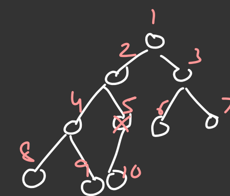
last non-leaf node in level order → $\left(\frac{N}{2}\right)$



$idx = \frac{5}{2} = 2$



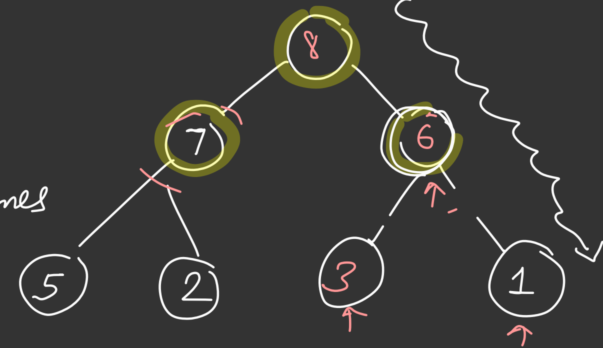
$idx = \frac{6}{2} = 3$



$idx = \frac{10}{2} = 5$

0	1	2	3	4	5	6	7
$\boxed{X}$	3	5	1	7	2	6	8

idx $\boxed{3, 2, 1}$



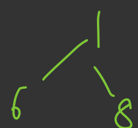
Code

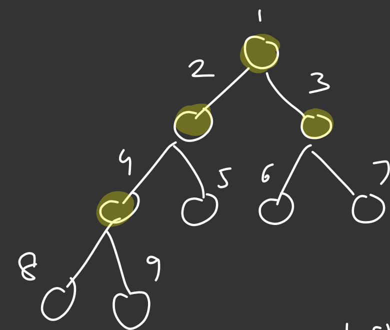
for(i = $\frac{n}{2}$; i>=1 ; i=i-1){ $\frac{N}{2}$ times

heapify (arr, i) ⟸ Not constant

Shifting down the node iteratively till it reaches the right position

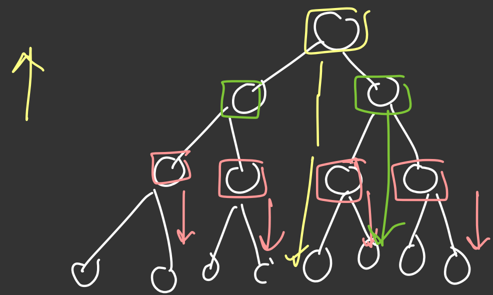
}





N = 9

last non leaf node $= \frac{9}{2} = 4$



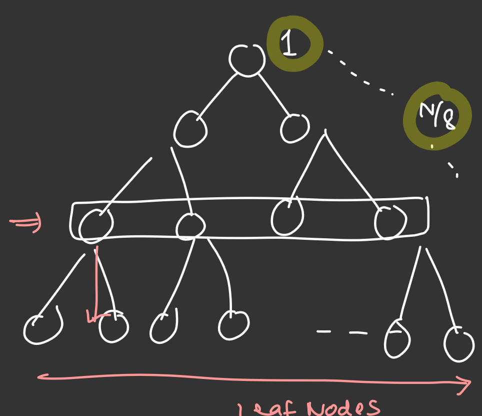
1 Node × 3 units for heapify

2 Nodes × 2 units for heapify

4 Nodes × 1 unit for heapify

Nodes ↓ Time for heapify ↑ $= O(N)$

PROOF



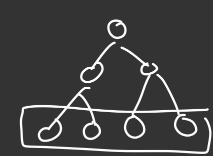
3 units

× 2 units

$= \frac{N}{4}$ × 1 unit ←

$\Rightarrow N/2$ × 0 units

N = 7



Total time = $\sum$ No of Nodes on each Level $\times$ Time at Level

$$= \frac{N}{4}(1) + \frac{N}{8}(2) + \frac{N}{16}(3) + \cdots\cdots$$

$$= \frac{N}{4}\left(\underbrace{\left(\frac{1}{1} + \frac{2}{2} + \frac{3}{4} + \cdots\right)}_{\text{AGP (AP + GP) both}}\right) = \frac{N}{\cancel{4}}(\cancel{4}) = \boxed{O(N)}$$

finite terms $\Rightarrow 4$

Sum of AGP

$$S = \frac{1}{1} + \frac{2}{2} + \frac{3}{4} + \frac{4}{8} + \cdots\cdots \infty$$

[infinite terms $r = \frac{1}{2}$ (converge)]

$$\frac{S}{2} = \frac{1}{2} + \frac{2}{4} + \frac{3}{8} + \frac{4}{16} + \cdots\cdots \infty$$

(Multiply both sides by $\frac{1}{2}$)

$-$

$$S - \frac{S}{2} = 1 + \left(\frac{2}{2} - \frac{1}{2}\right) + \left(\frac{3}{4} - \frac{2}{4}\right) + \left(\frac{4}{8} - \frac{3}{8}\right) + \cdots\cdots$$

$$\frac{S}{2} = 1 + \left(\frac{1}{2}\right) + \left(\frac{1}{4}\right) + \left(\frac{1}{8}\right) + \cdots\cdots \infty$$

$\boxed{GP}$

$$S = \frac{a}{1-r}$$

$$= \frac{1}{1-1/2}$$

$$\frac{S}{2} = 2$$

$$\Rightarrow \boxed{S = 4}$$

✓ Any array can be converted into heap array in-place in $O(N)$

(No need to use any additional PQ)

Sorting ⇒ Heap Sort

↳ use Priority Queue (Algo-1)

↳ In place (Algo-2)

Algo-1

	0	1	2	3	4	5	6	7
arr →	$\boxed{\times}$	3	5	1	7	2	6	8

~~3~~, ~~5~~, ~~1~~, ~~7~~, ~~2~~,
~~6~~, ~~8~~, 8

Min Heap
(library / Additional Data Structure)

→ Push all elements in PQ O(N log N)

→ Remove all elements from PQ & put it back.
O(N log N)

arr → | 1 | 2 | 3 | 5 | 6 | 7 | 8 |

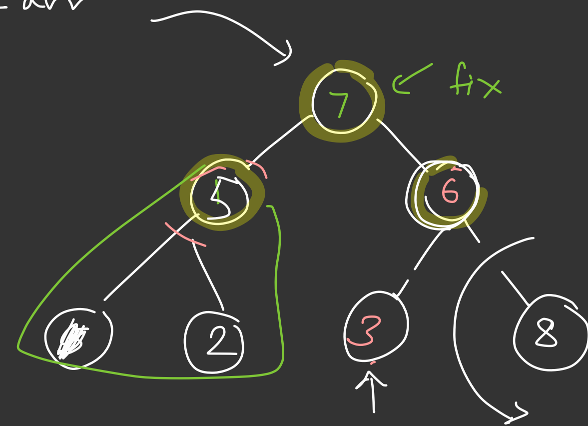
⇒ O(N log N) time

⇒ O(N) space by priority queue data structure

(Additional)

Algo-2

0	1	2	3	4	5	6	7	
$\boxed{X}$	3	5	1	7	2	6	8	← arr



⇒ Make a Max Heap

Code (A)

O(N)

```
for( i = n/2 ; i>=1 ; i=i-1){

    heapify(arr, i)

}
```

	1	2	3	4	5	6	7	
X	8	7	6	5	2	3	1	← arr

Size

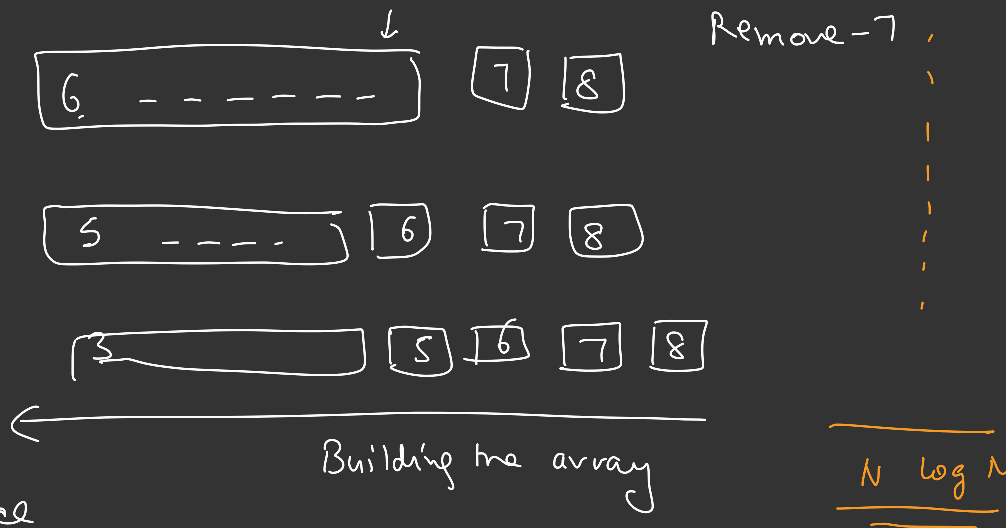
⇒ remove (N-1) elements from this array using Remove logic

7	5	6	1	2	3	$\boxed{8}$

Heapify

Size

Remove-8 (log N)



Heap Sort Inplace

Heap Sort

1. Build a heap in place $O(N)$ ⇒ code A
2. Remove N elements of heap $O(N \log N)$ ⇒ code B

TC: $O(N \log N)$

SC: $O(1)$

Code (B)

0	1	2	3	4	5	6	8
	1	7	6	5	2	3	8

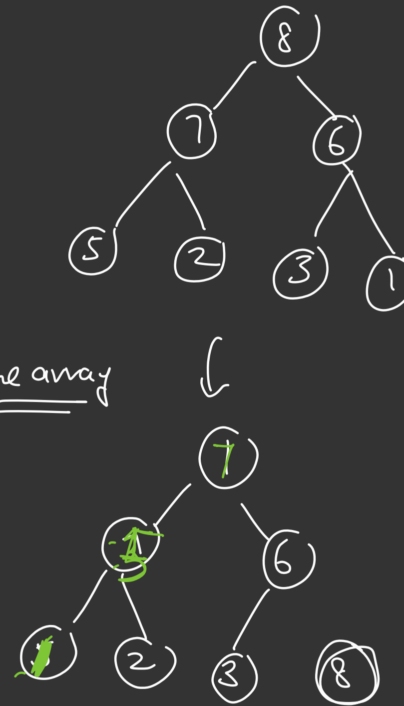
size ~~7~~ ~~8~~
7

N - 1
times

```
while ( size > 1 ) {

    swap ( Arr[1], Arr[size] )      ← same array
    size = size - 1
    heapify ( arr, 1 )              ← next largest comes to top

3
```



Each iteration is putting a larger element at back of array

ToDo.

void $\boxed{\text{heapify}}$ (arr, i) {

last class →

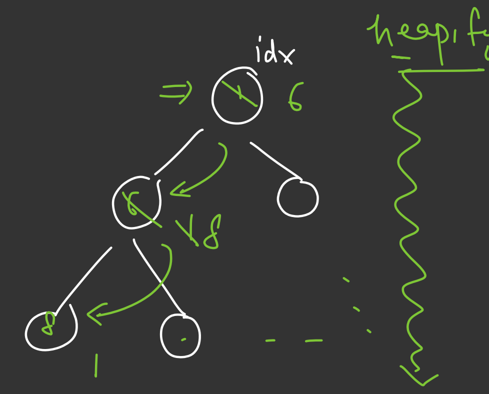
L = 2i

R = 2i + 1

Swapping

go Down to fix that part

}

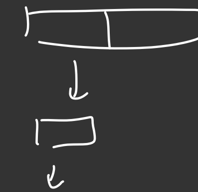


Dec

Max Heap

Inc array

Merge Sort	$O(N\log N)$	$O(N)$
Quick Sort	$O(N\log N)$ avg	$O(\log N)$
Heap Sort (inplace)	$O(N\log N)$	$O(1)$



BREAK . 10 25

PROBLEMS-

Q) N students, Top K performers based upon marks

marks = [95, 86, 322, 440, 56, 79, 152]

$K = 3$

$\boxed{K < N}$

440, 322, 152,

Brute Force Sort( ) and select last K $O(N \log N)$

Heap Algo1

1. Build a heap $O(N)$
2. Remove K from heap elements $O(K \log N)$

$= \underbrace{O(N + K \log N)}$ better

Heap Algo-2

Build a Min heap of size K & use it to maintain top K-element

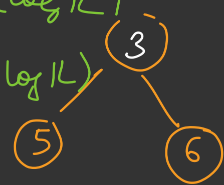
$K = 3$

$\underbrace{5,\ 6,\ \cancel{7}}_{K},\ \overset{K+1}{3},\ 8,\ 10,\ 12,\ 4$

at each step

→ 1 insertion $O(\log K)$

1 deletion $O(\log K)$



$K \log K$

$O(N \log K)$

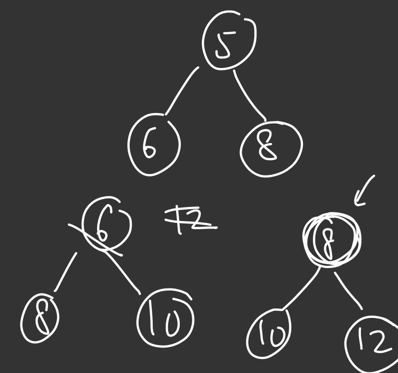
$(5, 6, \cancel{7})$ $\boxed{3}$

$(5, 6, \cancel{3})$ (8)

$(\cancel{5}, 6, 8)$

$(\cancel{6}, 8, 10)$ 12

$(8, 10, 12)$ $\boxed{4}$



$\Rightarrow \quad O(N + K \cdot \log N) \quad$ vs $\quad O(N \log K)$

Space $\Rightarrow$ $O(N)$ Space $= O(K)$

Running Median

Stream of integers ...

Print the median after every input

↓

Middlemost element (after sorting)

8, 2, 10, 1, 6, 12 →

(1, 2, $\boxed{6, 8}$, 10, 12)

odd → middle even → avg of 2 middle elements Avg = 7

8, 2, 10, 1, 6, 12 . . . .

Median 8, 5, 8, 5, 6, 7 . . . .

maintain a sorted array

↳ (2, 8)

↳ (1, 2, 8, 10)

↳ (1, 2, $\boxed{6, 8}$, 10, 12)

Brute ~~force~~

⇒ maintain a sorted array

⇒ for new insert, put the element in correct position } O(N) for each ins

= $O(N^2)$ → bad :(

12, 2, 8, 6, 1, 10

median

1, 2, 6, 8, 10, 12

$\boxed{6}$

9, 14

Shift Right

-3, -2, 1, 2, 6, ~~8~~, 9, 10, 12, 14, 20

prev just lower element

$\boxed{m}$

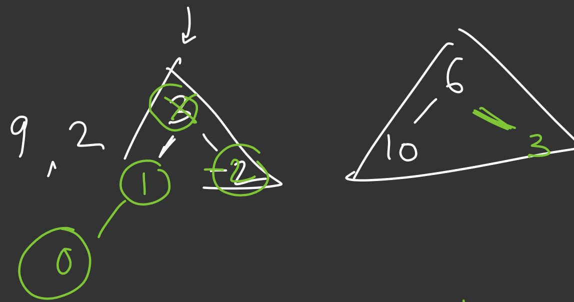
next higher element

Largest of all elemnts less than

↳ lowest element greater than median [MinHeap] median

[Max Heap]

3 ⇒ -2, 0, $\boxed{1, 2, 3}$, 6, 10



3, 6, 10, 0, -2, 1, 9, 2

med 3, 3, 6

Median = 3

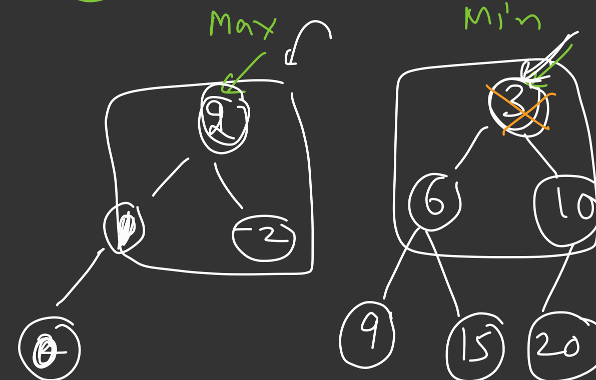
Median = 3,

Median = 6

median = 3

median = 3 K 3

Median = 1 9 > 1



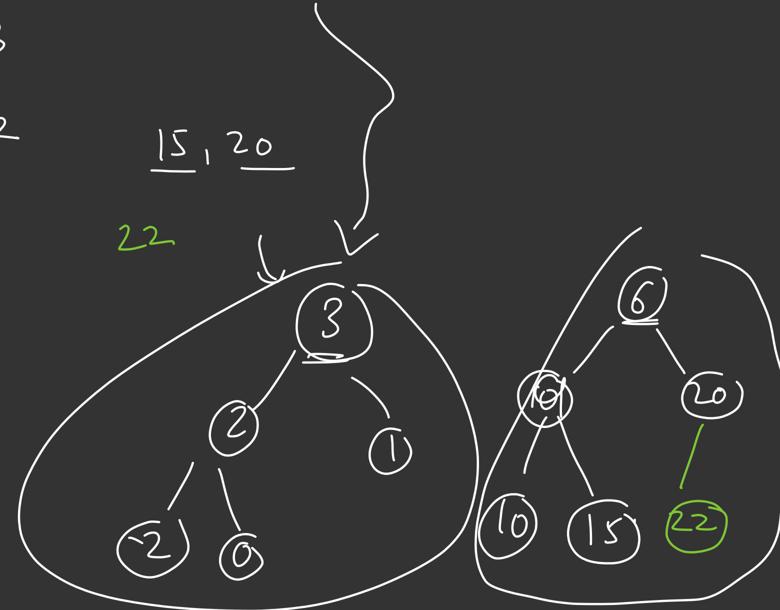
x x + 2

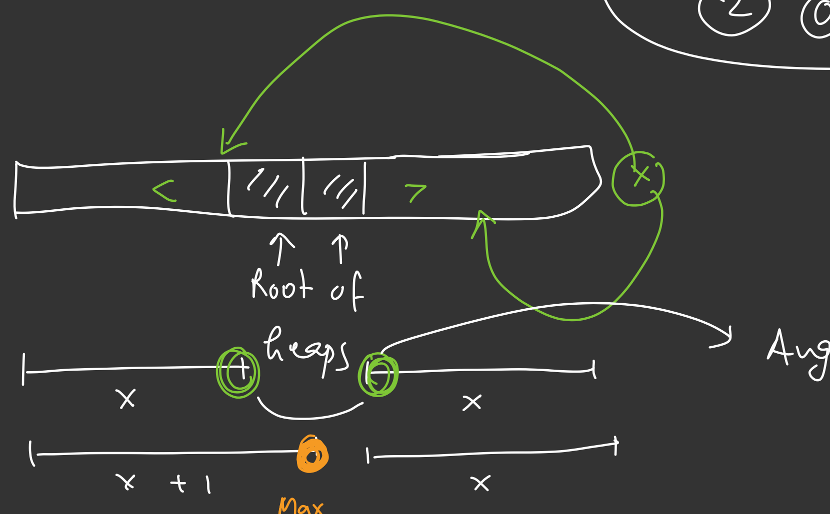
Median = 3 $\quad 2 < 3$

Median = 2 $\quad 15 > 2$

Median = 3

Median = 6

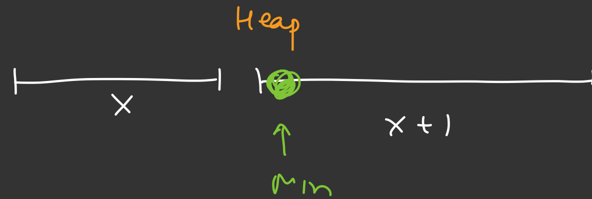




• Insertion

↳ Direct Insert

x , x → O(logN)



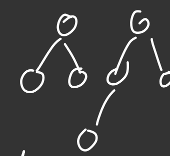
↳ Shift an element $H1 \leftrightarrow H2$ after insert

$O(\log N)$

- Insert $O(\log N)$
- Query → Root of Big heap

  ↑ $O(1)$

  → Avg of Roots equal heap



X , x + 2

x + 1 , x + 1

[_ _ _ _ _ _ _ .] $\Rightarrow O(N \log N)$

↓

better $O(N^2)$

Summary

Concepts

1. Array → Heap in O(N)
2. Array → sorted array (Heap sort) O(N log N)

Use cases

3. Top "K elements" out of N
4. Running median (2 Heaps)